FVPP: SAT-rješavači

Algoritmi DPLL, GRASP, Chaff i MiniSAT
Primjena u provjeri modela.

Pripremio: izv. prof. dr. sc. Alan Jović

Ak. god. 2022./2023.

Sadržaj

- SAT-problem
- Temeljni algoritam DPLL
- Algoritam GRASP
- Algoritmi Chaff i MiniSAT
- Primjena SAT-rješavača u FV

SAT-PROBLEM

SAT-problem

- Temeljni NP-kompletan problem
- Traži se **model** (interpretacija koja se evaluira u istinu) skupa formula
- Skup formula zadan najčešće u **CNF-obliku** (konjunkcija više klauzula):

$$\Gamma = K_1 \wedge K_2 \wedge \ldots \wedge K_m$$
$$= (k_{11} \vee \ldots \vee k_{1p}) \wedge (k_{21} \vee \ldots \vee k_{2r}) \wedge \ldots \wedge (k_{m1} \vee \ldots \vee k_{ms})$$

- K_i – *i*-ta klauzula; k_{ij} – *j*-ti literal u *i*-toj klauzuli
- Složenost SAT-problema za 3CNF ili više je eksponencijalna($O(2^n)$) u ovisnosti o broju atomičkih simbola (varijabli)

SAT-problem

- Iscrpna procedura rješavanja **CNF** SAT-problema:
  - Sustavno se pridjeljuju istinitosne vrijednosti atomičkim propozicijskim simbolima sve dok se ne nađe model ili iscrpe sve mogućnosti.
  - Grananje na svakom atomičkom simbolu – istina ili laž
  - Ako se ne nađe model (pronađe se konflikt), vraća se na prethodnu varijablu i isprobava se drugu mogućnost
  - **Za *n* atoma: 2^n pridruživanja**.
- **Najgori slučaj: $O(2^n)$**.
- **Prosječni slučaj (suvremeni rješavači): $O(n^p)$.**
- Primjeri heurističkih SAT-rješavača koji garantiraju završetak pretrage:
  - DPLL, Grasp, Chaff (zChaff), MiniSAT, SatZ...

Zašto se bavimo SAT-rješavačima?

- Problemi u mnogim područjima mogu se svesti na SAT-problem
  - **Formalna verifikacija sklopovlja i programa** ← **fokus**
  - Optimizacijski problemi
  - Umjetna inteligencija (npr. dijagnoza kvarova), automatsko planiranje (*satplan*)
- Drugi suvremeni rješavači (npr. QBF, SMT) posuđuju tehnike korištene u SAT-rješavačima
- SAT-rješavači su još uvijek aktivno područje istraživanja, natjecanja se kontinuirano održavaju:

  http://www.satcompetition.org/

TEMELJNI ALGORITAM DPLL

DPLL

- Davis-Putnam-Logemann-Loveland, 1962.
- Predstavlja osnovu za sve suvremene SAT-rješavače
- U svakom koraku algoritma koriste se dvije heuristike:
  - **Propagacija jedinične klauzule** (engl. *Unit clause propagation*, dalje: PJK)
  - **Uklanjanje klauzula s čistim literalima** (engl. *Pure literal elimination*, dalje: UKČL)

Propagacija jedinične klauzule

- Ako je neka klauzula jedinična, **što znači da sadrži samo jedan literal,** tada se varijabli može pridružiti onu vrijednost koja čini tu klauzulu istinitom
- Takva jedinična klauzula se eliminira iz daljnjeg razmatranja, a sve one klauzule koje sadrže tu varijablu se razrješavaju
- PJK se ponavlja sve dok više nema jediničnih klauzula
- Smanjivanje broja klauzula se može u pojedinim specifičnim slučajevima svesti na kaskadu kojom se može vrlo brzo doći do modela

Propagacija jedinične klauzule

- Primjer:

$$\Gamma = ((x1 \vee x2) \wedge (\neg\, x1 \vee \neg\, x2) \wedge (x1))$$
$$= ((T \vee x2) \wedge (F \vee \neg\, x2) \wedge (T))$$
$$= ((T) \wedge (\neg\, x2) \wedge (T))$$
$$= (\neg\, x2)$$
$$= T$$

Jedinična klauzula: x1, zatim $\neg$ x2

Rješenje: {x1=T, x2=F}

Uklanjanje klauzula s čistim literalima

- **Čisti literal** je varijabla koja se pojavljuje u samo **jednoj polarnosti** (negirana ili nenegirana) u **svim** klauzulama gdje je prisutna
- **Čistom literalu uvijek se može pridijeliti takvu vrijednost da klauzula postane istinita i da time takva klauzula prestane biti relevantna u potrazi za modelom**
- Sve takve klauzule se uklanjaju i čisti literali se pamte, što se ponavlja dokle je to moguće
- Primjer:

$\Gamma = ((x1 \lor x2 \lor x3) \land (\neg x1 \lor \neg x2 \lor x3) \land (\neg x1 \lor \neg x2) \land (x1 \lor \neg x2))$

x3 je čisti literal, klauzule koje ga sadrže zanemaruju se i pamti se $(x3 = T)$. Ostaje:

$= ((\neg x1 \lor \neg x2) \land (x1 \lor \neg x2))$

$\neg$x2 je čisti literal, zanemaruju se obje klauzule i pamti se $(x2 = F)$.

$= T$

Moguće rješenje: {x1=T, x2 =F, x3=T}   - x1 može biti i F, svejedno je

DPLL

- Algoritam započinje s formulom u CNF obliku
- Algoritam u svakom koraku koristi najprije PJK dokle je moguće, a zatim UKČL dokle je moguće
- Ako nije pronađeno rješenje, treba se zatim odabrati varijablu za grananje
- Učinkovitost DPLL-a značajno ovisi o **strategiji izbora varijable za grananje**, osnovna strategija je **slučajni** izbor varijable i slučajni izbor njene vrijednosti (T ili F)
- Algoritam završava ako:
  - postoji konzistentan skup literala (našao model) ili
  - ispitane su sve mogućnosti i svugdje se naišlo na konflikt (nekonzistentan skup literala)

Osnovna strategija pretraživanja s povratkom na prethodnu varijablu

1. Donesi odluku o varijabli grananja i pridruži joj vrijednost
2. Zaključi implicirana pridruživanja deduktivnim procesom (PJK i UKČL).
   - Može dovesti do nekonzistentnih klauzula: **konflikt!**
   - Pridruživanje koje je dovelo do konflikta zove se konfliktno pridruživanje.
3. Ako dođe do konflikta, pokušava se pridruživanje alternativne vrijednosti varijabli grananja (kronološki povrat, tzv. *backtracking*)
4. Ako opet dođe do konflikta, vraća se slijedno nazad uz hijerarhiju grananja i provode se pridruživanja alternativne vrijednosti varijabli grananja
5. Ako su isprobane sve mogućnosti varijabli grananja i svugdje je došlo do konflikta, onda problem nije zadovoljiv

Poboljšanja DPLL-a

- Poboljšanja DPLL-a išla su u smjeru:
  - Uvođenja raznih heuristika pri izboru literala za grananje
  - Variranja osnovne strategije povratka na prethodnu varijablu pri konfliktu
  - Optimiranja propagacije ograničenja u klauzulama
  - Optimiranja struktura podataka koje se koriste za implementaciju

ALGORITAM GRASP

GRASP

- **GRASP** - Generalized seaRch Algorithm for the Satisfiability Problem (Silva, Sakallah, 1996.)
- Koristi se heuristika za varijablu grananja: pridijeli svakoj varijabli obje mogućnosti, a zatim odredi koje će pridruživanje zadovoljiti najviše klauzula (2*n* pridjeljivanja * provjera *p* klauzula)
  - Time se ustvari pronalazi najčešći literal
- Značajke algoritma:
  - **Učenje novih klauzula kroz konflikte** (engl. *conflict-driven clause learning*, CDCL).
  - **Grafovi implikacija** za propagaciju jediničnih klauzula i analizu konflikata.
  - Analiza konflikata i grafovi implikacije dovode do **ne-kronološkog** povrata na prethodnu varijablu (tzv. *backjumping*)!

Grafovi implikacija

- PJK je specijalan slučaj propagacije Booleovih ograničenja (engl. *Boolean Constraint Propagation*, PBO)
- GRASP radi PBO koristeći **grafove implikacija** (engl. *implication graph*)
  - npr. $K_i = (x \vee \neg y)$, ako je pri grananju $y = 1$ tada je $x = 1$ **impliciran** ili **forsiran**, zato što je nužno da bude $x = 1$ da bi formula bila SAT
- PBO je iterativna primjena implikacija sve dok to više nije moguće ili je došlo do konflikta

Grafovi implikacija

- **Pridruživanje prethodnika** varijable x, u oznaci A(*x*) je pridruživanje vrijednosti 0 svim drugim literalima osim x u toj klauzuli, čime je x forsiran na vrijednost 1.
- Primjer
  - $K_j = (x \vee y \vee \neg z)$,
    $A(x) = \{y{:}0, z{:}1\}$, $A(y) = \{x{:}0, z{:}1\}$, $A(z) = \{x{:}0, y{:}0\}$
  - Pridruživanje prethodnika direktno "prokazuje" vrijednosti varijabli koje su prouzročile forsiranje varijable "x" na 1.

Grafovi implikacija

- Grafovi implikacija prikazuju **odluke o grananju** i **implicirana pridruživanja**.
- Prethodnici čvora *x* u grafu implikacija odgovaraju pridruživanju prethodnika A(*x*), a povezani su s čvorom *x* lukovima na kojima je označena jedinična klauzula (implikacija) koja je dovela do *x*.
  - U grafu implikacija ne prikazuju se prethodnici čvora ako je čvor *x* odluka o grananju!
- Za specijalan konfliktni čvor κ (kappa), pridruživanje prethodnika $A(\kappa)$ je pridruživanje varijabli u klauzuli koja je nekonzistentna i time neistinita.

Primjer grafa implikacija

Primjer prilagođen od: W. Klieber, CMU, 2011.

Oznaka razine odluke. Neka pridruživanja bile su grananja, a neke implikacije (forsiranja)

$K_1 = (\neg x_1 \vee x_2)$

$K_2 = (\neg x_1 \vee x_3 \vee x_9)$

$K_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$K_4 = (\neg x_4 \vee x_5 \vee x_{10})$

$K_5 = (\neg x_4 \vee x_6 \vee x_{11})$

$K_6 = (\neg x_5 \vee \neg x_6)$

$K_7 = (x_1 \vee x_7 \vee \neg x_{12})$

$K_8 = (x_1 \vee x_8)$

$K_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$

...

- Trenutačno pridruživanje istinitosti:

  $\{x_9=0@1,\ x_{12}=1@2,\ x_{13}=1@2,\ x_{10}=0@3,\ x_{11}=0@3\ ,\ \ldots\}$
- Trenutačna odluka pri pridruživanju: $\{x_1=1@6\}$
- **Ideja:** pronaći klauzulu koja, uz trenutačnu odluku pri pridruživanju i ostalih trenutnih pridruživanja istinitosti, može dovesti do impliciranja (forsiranja) određenog literala u toj klauzuli. Nakon toga, nacrtati pridruživanja kao čvorove koji su doveli do impliciranja literala i povezati ih s čvorom impliciranog literala strelicom s oznakom klauzule. Postupak nastaviti uzimajući u obzir implicirane literale dok se ne dođe do konflikta, do rješenja problema ili do potrebe za grananjem.

Napomena: i klauzula i varijabli u trenutačnom pridruživanju ima još, ali nam nisu bitni za rješenje ovog primjera (primijetiti tri točke ...)

Primjer grafa implikacija

Primjer prilagođen od: W. Klieber, CMU, 2011.

Oznaka razine odluke. Neka pridruživanja bile su grananja, a neke implikacije (forsiranja)

- Trenutačno pridruživanje istinitosti:

  $\{x_9=0@1,\ x_{12}=1@2,\ x_{13}=1@2,\ x_{10}=0@3,\ x_{11}=0@3\ ,\ \ldots\}$

- Trenutačna odluka pri pridruživanju: $\{x_1=1@6\}$

$K_1 = (\neg x_1 \vee x_2)$

$K_2 = (\neg x_1 \vee x_3 \vee x_9)$

$K_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$K_4 = (\neg x_4 \vee x_5 \vee x_{10})$

$K_5 = (\neg x_4 \vee x_6 \vee x_{11})$

$K_6 = (\neg x_5 \vee \neg x_6)$

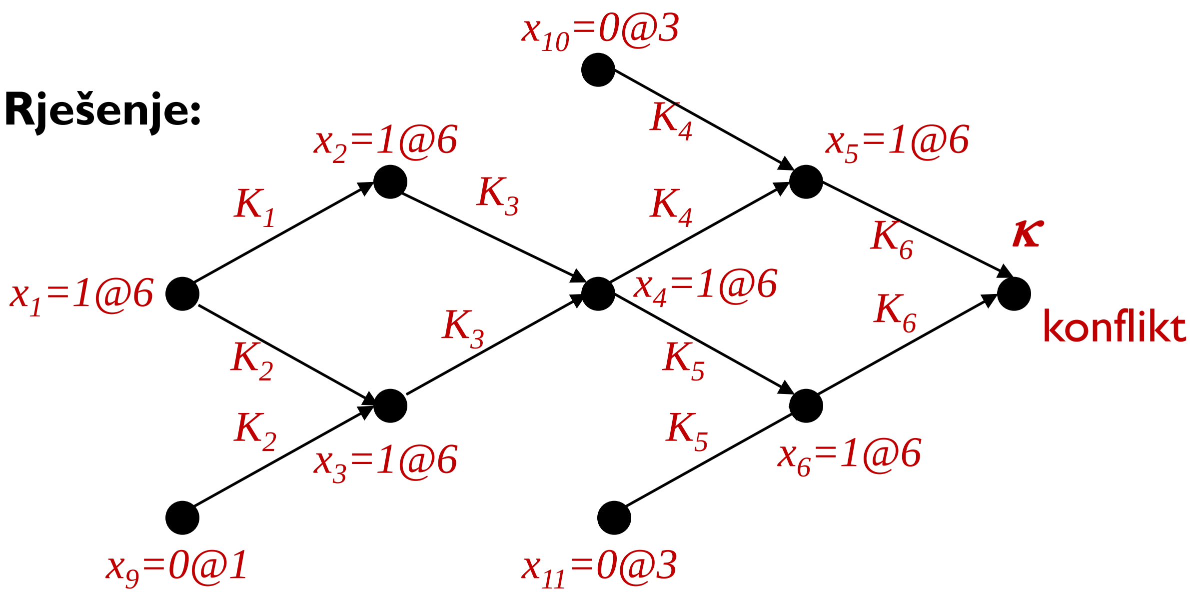
$K_7 = (x_1 \vee x_7 \vee \neg x_{12})$

$K_8 = (x_1 \vee x_8)$

$K_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$

...

Rješenje:



Napomena: i klauzula i varijabli u trenutačnom pridruživanju ima još, ali nam nisu bitni za rješenje ovog primjera (primijetiti tri točke ...)

Analiza konflikta

- Ako se dogodi konflikt, analizira se graf implikacija:
  - Dodaje se nova klauzula koja će spriječiti istraživanje istog konflikta u budućnosti
    $\Rightarrow$ **Učenje novih klauzula kroz konflikte (CDCL):**
    - Neka je KP konfliktno pridruživanje klauzule, u našem primjeru:.

      KP = $\{x_1=1@6,\ x_9=0@1,\ x_{10}=0@3,\ x_{11}=0@3\}$

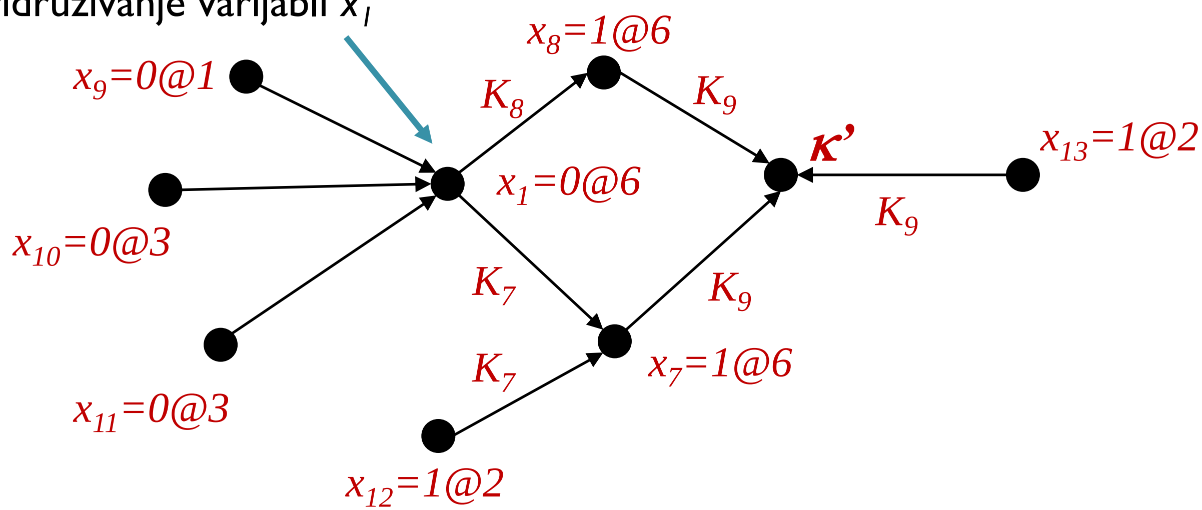
      Naučena nova klauzula koja će izbjeći isti konflikt u budućnosti jednaka je **negaciji konfliktnog pridruživanja**:

      $K_{new}=\neg KP = \neg(x_1 \wedge \neg x_9 \wedge \neg x_{10} \wedge \neg x_{11}) = \neg x_1 \vee x_9 \vee x_{10} \vee x_{11}$
  - Odredi se razina odluke do koje se treba vratiti (na koju treba skočiti), to ne mora nužno biti prethodna razina
    $\Rightarrow$ **Ne-kronološki povratak**

Analiza konflikta

- Literal $l \in$ KP je jedinstvena implikacijska točka (JIT) akko svaki drugi literal $m \in$ KP , $m \neq l$ ima raniju razinu odluke od l.
- Do JIT se dolazi uklanjanjem svih implikanata u implikacijskom grafu (konfliktnom pridruživanju KP) na zadnjoj razini odluke (u našem primjeru uklone se x_5, x_4,...) sve dok se ne dođe do zadnjeg ($x_1 = 1@6$.) U našem slučaju, literal $x_1=1$ je JIT
- Ako bismo sada zbog nastalog konflikta pridružili $x_1= 0$, izgradnjom novog grafa implikacija pokazuje se da bi se ponovno dobio konflikt, κ', a ovaj put KP' = $\{x_9=0@1, x_{10}=0@3, x_{11}=0@3, x_{12}=1@2, x_{13}=1@2\}$:

x_9, x_{10} i x_{11} su uzrokovali alternativno pridruživanje varijabli x_1



Ne-kronološki povratak

- Ideja ne-kronološkog povratka (*backjumping*) je vratiti se na **zadnju (najdublju) razinu odluke** koja postoji kod literala unutar **KP'** (dakle, nakon što smo isprobali alternativnu vrijednost varijable grananja).
- Kod kronološkog povratka ta razina bi bila razina od JIT – 1, budući da smo isprobali obje vrijednosti varijable na razini JIT. S druge strane, *backjumping* nas u ovom slučaju dovodi na razinu odluke 3

- Na taj način drastično se smanjuje pregledavanje prostora stanja na razinama 4 i 5, jer bi i na tim razinama uvijek dolazilo do konflikta, budući da KP' ne ovisi izravno o pridruživanjima na tim razinama (klauzula izvedena i naučena iz KP': $\neg$KP' = $x_9 \vee x_{10} \vee x_{11} \vee \neg x_{12} \vee \neg x_{13}$ bi bila uvijek nezadovoljena na razinama 4 i 5)

Problemi GRASP-a

- Rast broja klauzula s brojem ne-kronoloških povrata (u općem slučaju eksponencijalno)
  - Rješenje: ograničava se veličina novo naučene klauzule na *k* literala
- Veliki fiksni trošak (engl. *overhead*) prilikom analize konflikta i prilikom odluke o varijabli grananja (svaki put treba tražiti najčešći literal)
- Heuristika koju GRASP koristi pri odabiru varijable grananja ne daje uvijek najbolje rezultate

Okvirni pseudokod GRASP-a

```
Neka je Γ formula u CNF-obliku
TrenutnoPridr = {};
while (true) {
    while (vrijednost od Γ za TrenutnoPridr nepoznata){
        OdlučiGrananje (); // Dodaj literal u TrenutnoPridr
        PBO(); // Dodaj implicirane literale u TrenutnoPridr
    }
    if (TrenutnoPridr zadovoljava Γ) {return true;}
    AnalizaKonfliktaIUčenjeNoveKlauzule();
    if (naučena klauzula je prazna) {return false;}
    PovratakNaPrethodnuVarijablu();
    PBO();
}
```

ALGORITMI CHAFF I MINISAT

Chaff

- Moskewicz et al., 2001. (suradnja UC Berkeley, MIT i Princeton University)
- Još uvijek vrhunski SAT-rješavač, poboljšavan do 2007., ugrađen u NuSMV i NuXmv (zChaff)
- Koristi neke tehnike od prethodnika (DPLL, GRASP)
- Fokus na boljem inženjerstvu svih aspekata pretrage
- Naglasak na **optimizaciji propagacije Booleovih ograničenja** (jer tu algoritmi potroše najviše, oko 80% vremena, samo oko 10% na analizi konflikta) i smanjenju fiksnog troška pri odluci o varijabli grananja

Chaff

- Značajke Chaffa
  - Učinkovita PBO
    - **Promatraju se dva literala**
    - Brzi povratak na prethodnu razinu
  - Učinkovita procedura odlučivanja o grananju
    - Lokalizira prostor pretraživanja
  - CDCL kao kod GRASP-a
  - Uklanjanje naučenih klauzula
  - Pretraga iznova (*restart*)

Učinkovita PBO

- Što određuje forsiranje vrijednosti (implikaciju) literala?
  - Svi literali u klauzuli osim jednoga iznose 0
  - Npr. za $(v1 \vee v2 \vee v3)$ implikacije su: $(0 \vee 0 \vee v3)$ ili $(0 \vee v2 \vee 0)$ ili $(v1 \vee 0 \vee 0)$
  - Za klauzulu sa p literala, forsiranje literala se događa nakon što $p - 1$ literala imaju pridruženu vrijednost *false*
  - Ideja: mogli bismo potpuno ignorirati analiziranje prvih $p - 2$ pridruživanja toj klauzuli i reagirati tek kad se prijeđe s $(p - 2)$. na $(p - 1)$. pridruživanje
  - Praktično: za svaku klauzulu odaberu se **dva literala** na slučajan način koji će se "promatrati", a ignoriraju se pridruživanja svim ostalim literalima u toj klauzuli

Primjer

- U klauzuli $(v1 \vee v2 \vee v3 \vee v4 \vee v5)$ promatramo v1 i v2:
- (**v1=X** $\vee$ **v2=X** $\vee$ v3 = ? $\vee$ v4=? $\vee$ v5=?), pridruživanja za v3,v4,v5 ne razmatramo u toj klauzuli

Primjer učinkovite PBO

Primjer prilagođen od T. Heyman i W. Klieber, CMU

$K1 = \underline{v2} \lor \underline{v3} \lor v1 \lor v4$

$K2 = \underline{v1} \lor \underline{v2} \lor \neg v3$

$K3 = \underline{v1} \lor \neg \underline{v2}$

$K4 = \neg \underline{v1} \lor \underline{v4}$

$K5 = \neg v1$

Početni slučajno odabrani promatrani literali

Primjer učinkovite PBO

Trenutačni stog pridruživanja = {v1 = F} (zbog implikacije na jediničnoj klauzuli K5)

$K1 = \underline{v2} \lor \underline{v3} \lor \mathbf{v1} \lor v4$

$K2 = \underline{\mathbf{v1}} \lor \underline{v2} \lor \neg v3$

$K3 = \underline{\mathbf{v1}} \lor \neg\, \underline{v2}$

$K4 = \neg\, \underline{\mathbf{v1}} \lor \underline{v4}$

1. Razmatraju se sve klauzule gdje odabrani literal daje vrijednost F i time smanjuje veličinu klauzule
2. Ne razmatraju se one klauzule gdje bi odabrani literal dao vrijednost T, jer je time cijela klauzula T
3. Ne razmatraju se one klauzule u kojima se odabrani literal ne promatra

Primjer učinkovite PBO

Trenutačni stog pridruživanja = {v1 = F}

$K1 = \underline{v2} \lor \underline{v3} \lor v1 \lor v4$

$K2 = v1 \lor \underline{v2} \lor \underline{\neg v3}$

$K3 = v1 \lor \neg \underline{v2}$

$K4 = \neg \underline{v1} \lor \underline{v4}$

Razmatramo 2. i 3. klauzulu.

U 2. klauzuli odabiremo $\underline{\neg v3}$ kao sljedeći promatrani literal
3. klauzula postala je ovim pridruživanjem jedinična klauzula. Forsirani literal (implikaciju) v2 = F dodajemo u red pridruživanja koja moramo obraditi

Primjer učinkovite PBO

Trenutačni stog pridruživanja = {v1 = F, v2 = F}

$K1 = v2 \lor \underline{v3} \lor v1 \lor \underline{v4}$

$K2 = v1 \lor v2 \lor \underline{\neg v3}$

$K3 = v1 \lor \neg \underline{v2}$

$K4 = \neg \underline{v1} \lor \underline{v4}$

Razmatramo 1. i 2. klauzulu, gdje odabrani literal v2 = F smanjuje veličinu klauzule.

U 1. klauzuli odabiremo v4 kao sljedeći promatrani literal
2. klauzula postala je jedinična klauzula. Implikaciju v3 = F dodajemo u red pridruživanja koja moramo obraditi

Primjer učinkovite PBO

Trenutačni stog pridruživanja = {v1 = F, v2 = F, v3 = F}

$K1 = v2 \lor v3 \lor v1 \lor \underline{v4}$

$K2 = v1 \lor v2 \lor \underline{\neg v3}$

$K3 = v1 \lor \neg \underline{v2}$

$K4 = \neg \underline{v1} \lor \underline{v4}$

1. klauzula se razmatra, gdje odabrani literal v3 = F smanjuje veličinu klauzule.

1. klauzula postala je jedinična klauzula. Implikaciju v4 = T dodajemo u red pridruživanja koja moramo obraditi.

Primjer učinkovite PBO

Trenutačni stog pridruživanja = {v1 = F, v2 = F, v3 = F, v4 = T}

$K1 = v2 \lor v3 \lor v1 \lor \underline{v4}$

$K2 = v1 \lor v2 \lor \underline{\neg v3}$

$K3 = v1 \lor \neg \underline{v2}$

$K4 = \neg \underline{v1} \lor \underline{v4}$

Budući da nema konflikta i nemamo više literala u redu obrade, PBO završava i treba se donijeti odluka o sljedećem grananju. Nema više nepridijeljenih vrijednosti. Pronađen je model i procedura je završila.

Sažetak PBO-a

- Ako dođe do konflikta:
  - Samo se odmota stog pridruživanja: Novi trenutačni stog pridruživanja = {poč. stanje stoga pridruživanja prije odluke o grananju}
  - Ne mijenjaju se promatrani literali (koji su vjerojatno mijenjani tijekom PBO) u bazi klauzula
  - Potrebno je samo promijeniti vrijednost varijable koja je dovela do konflikta kroz PBO, što se može napraviti u konstantnom vremenu
  - Nadalje, promjenom vrijednosti varijable koja je dovela do konflikta kroz PBO djelovat će samo na manji podskup klauzula od onoga u prvom pridruživanju, jer su **promatrani literali na nekim mjestima mijenjani**

- Ukupni je cilj minimizirati pristup klauzulama, jer je to pristupanje vremenski skupo

Heuristika odluke o varijabli grananja - **VSIDS**

- Engl. ***V**ariable **S**tate **I**ndependent **D**ecaying **S**um*
  - Varijable se rangiraju prema broju prisutnih literala (negiranih i nenegiranih) u **inicijalnoj** bazi klauzula
  - Ideja je da se literali koji se češće pojavljuju u novijim konfliktnim klauzulama najprije odabiru čime se lokalizira pretraga; a to se postiže:
    - Broj prisutnih literala za neku varijablu se inkrementira (za 1, tzv. *bump*) samo u slučaju dodavanja novih klauzula
    - Periodično, pomnože se svi brojevi s nekom konstantom $x < 1$ (raspad, engl. *decay*)
  - Pri odluci o grananju, odabire se prva po rangu (nepridijeljena) varijabla (i to njezin češći literal), a u slučaju izjednačenja, literal se odabere slučajno
  - **Odabrani literal postavlja se na FALSE radi smanjenja veličine klauzule**

VSIDS primjer

Inicijalna baza klauzula

$x1 \lor x4$
$x1 \lor \neg x3 \lor \neg x8$
$x1 \lor x8 \lor x12$
$x2 \lor x11$
$\neg x7 \lor \neg x3 \lor x9$
$\neg x7 \lor x8 \lor \neg x9$
$x7 \lor x8 \lor x10$

Rezultati:
4: x8
3: x1,x7
2: x3
1: x2,x4,x9,x10,x11,x12

Za grananje se odabere literal x8 (ne ¬x8), jer je najčešći (x8 – 3, ¬x8 – 1)

Dodana klauzula

$x1 \lor x4$
$x1 \lor \neg x3 \lor \neg x8$
$x1 \lor x8 \lor x12$
$x2 \lor x11$
$\neg x7 \lor \neg x3 \lor x9$
$\neg x7 \lor x8 \lor \neg x9$
$x7 \lor x8 \lor x10$
$x7 \lor x10 \lor \neg x12$

Rezultati:
4: x8, x7
3: x1
2: x3, x10, x12
1: x2,x4,x9,x11

Za grananje se opet odabere x8, jer je taj literal češći od x7 (3 : 2)

VSIDS primjer

Množenje s npr. 0.5 (raspad), zaokruži prema dolje

$x1 \vee x4$
$x1 \vee \neg x3 \vee \neg x8$
$x1 \vee x8 \vee x12$
$x2 \vee x11$
$\neg x7 \vee \neg x3 \vee x9$
$\neg x7 \vee x8 \vee \neg x9$
$x7 \vee x8 \vee x10$
$x7 \vee x10 \vee \neg x12$

Rezultati:
2: x8, x7
1: x3, x10, x12, x1
0: x2,x4,x9,x11

Za grananje se opet odabere x8, jer je taj literal češći od x7 (3 : 2)

Dodana nova klauzula

$x1 \vee x4$
$x1 \vee \neg x3 \vee \neg x8$
$x1 \vee x8 \vee x12$
$x2 \vee x11$
$\neg x7 \vee \neg x3 \vee x9$
$\neg x7 \vee x8 \vee \neg x9$
$x7 \vee x8 \vee x10$
$x7 \vee x10 \vee \neg x12$
$\neg x12 \vee x10$

Rezultati:
2: x8, x7, x10, x12
1: x3, x1
0: x2,x4,x9,x11

Sada bi se slučajno izabralo između x8 i x10 pri grananju (oba literala se pojavljuju 3 puta)

Interakcija PBO – VSIDS

- PBO obrađuje pridruživanja, čime neki literali prestaju biti promatrani
- Rang kod VSIDS-a se relativno sporo mijenja
- Općenito, što je varijabla više rangirana po VSIDS-u, to je vjerojatnije da više nije promatrana (jer ju je PBO zamijenio s nekom drugom)
- VSIDS utječe na to koje varijable će se naći u novo naučenim klauzulama proizašlima iz konflikta
- Bitno je zbog brzine da VSIDS održava strogo lokalizirane pretrage (pogotovo u slučaju velikog broja varijabli)

Brisanje naučenih klauzula

- Da bi se spriječio problem eksplozije zauzetog memorijskog prostora koji se može dogoditi učenjem novih klauzula nakon konflikta, uvodi se brisanje nekih naučenih klauzula
- Chaff koristi **planirano lijeno uklanjanje** (engl. *scheduled lazy deletion*)
  - **U trenutku dodavanja** klauzule, za nju se ocijeni kada (i da li uopće) će se izbrisati
  - Koristi se **faktor relevantnosti klauzule *N*** (tipično 100 – 200) koji kaže da kad prvi put više od *N* literala u klauzuli postane nepridijeljeno (odmatanjem stoga pridruživanja) da će se klauzula označiti za uklanjanje

Pretraga iznova (*restart*)

- Služi za rješavanje teških problema (bilo SAT, bilo ne-SAT)
- Napuštanje trenutačnog stanja pretrage i početak nove pretrage, uz zadržavanje naučenih (konfliktnih) klauzula da se ne bi ponovila analiza
- **Dodaje se mali šum pri prvih nekoliko odluka o grananju kako bi se istražio neki drugi put pretraživanja**
- I šum i učestalost pretrage iznova se mogu konfigurirati; po defaultu, učestalost se usporava sa svakom novom pretragom iznova

- Zbog pretrage iznova i brisanja nekih naučenih klauzula, teoretski se može dogoditi da pretraga ne bude kompletna. Kompletnost pretrage se ipak zadržava tako što se faktor relevantnosti klauzule *N* povećava s vremenom
- Kompletnost je uvijek održana ako se isključi pretraga iznova, no to može značajno produžiti računanje kod teških problema

Algoritam MiniSAT (N. Eén, N. Sörensson, 2003.-2008.)

- Minimalan ☺, dobro dokumentiran i učinkovit SAT-rješavač
- Manje razlike u odnosu na Chaff:
  - Najveći fokus na **učinkovitoj analizi konfliktnih klauzula**
    - Minimizira se veličina naučene klauzule detaljnijim razmatranjem klauzula koje su dovele do konflikta
    - Konfliktna klauzula se odmotava unazad primjenom **rezolucijskog pravila** na zadnjem propagiranom literalu
    - Staje se kada nova klauzula sadržava točno jedan novi literal zaključen od trenutka zadnje pretpostavke (grananja)
  - MiniSAT **ne razlikuje polarnosti literala** kod VSIDS mjere – uvijek pridjeljuje vrijednost *false* najviše rangiranoj varijabli

Algoritam MiniSAT (N. Eén, N. Sörensson, 2003.-2008.)

- Rezolucijsko pravilo (J. A. Robinson): Logička posljedica dviju istinitih, konjunkcijom vezanih, univerzalno kvantificiranih normaliziranih klauzula (disjunkcija literala) je normalizirana klauzula **bez jednog komplementarnog para literala.**

1. $(A_1 \lor A_2 \lor \ldots \lor A_n \lor \mathbf{B})$
2. $((\neg \mathbf{B}) \lor C_1 \lor C_2 \lor \ldots \lor C_p)$

---

3. $(A_1 \lor A_2 \lor \ldots \lor A_n \lor C_1 \lor C_2 \lor \ldots \lor C_p)$

Naprimjer, rezolucijsko pravilo nad varijablom a:

ako $(a \lor b)$ i $(\neg a \lor c \lor d)$ **onda zaključi** $(b \lor c \lor d)$

Primjer analize konflikta

Antecedent	Pridruživanje
pretpostavka	e=F
$\neg f \vee e$	f=F
$\neg g \vee f$	g=F
$\neg h \vee g$	h=F
pretpostavka	a=T (nova razina)
$b \vee \neg a \vee e$	b=T
$c \vee e \vee f$	c=T
$d \vee \neg b \vee h$	d=T

Npr. konflikt klauzule $d \vee \neg b \vee h$ s klauzulom: $\neg b \vee \neg c \vee \neg d$

Rezolucija 1 (po d):
$\neg b \vee \neg c \vee h$

Rezolucija 2 (po c):
$\neg b \vee h \vee e \vee f$

Tu stajemo, b je ostala jedina varijabla od trenutka zadnje pretpostavke (a=T), naučili smo novu konfliktnu klauzulu:
$\neg b \vee h \vee e \vee f$

Daljnja minimizacija veličine naučene klauzule

Antecedent	Pridruživanje
pretpostavka	e=F
$\neg f \vee e$	f=F
$\neg g \vee f$	g=F
$\neg h \vee g$	h=F
pretpostavka	a=T (nova razina)
$b \vee \neg a \vee e$	b=T
$c \vee e \vee f$	c=T
$d \vee \neg b \vee h$	d=T

Za daljnju minimizaciju veličine naučene klauzule može se "pohlepno" primijeniti rezolucijsko pravilo dokle ide, čak i iza razine posljednje pretpostavke

To MiniSAT ne radi

$\neg b \vee \mathbf{h} \vee e \vee f$

$\Downarrow$

$\neg b \vee \mathbf{g} \vee e \vee f$

$\Downarrow$

$\neg b \vee e \vee \mathbf{f}$

$\Downarrow$

$\neg b \vee e$

Gdje stati?
(nije uvijek manja naučena klauzula bolja)

Predobrada i unutarnja obrada

- U novije vrijeme veliki naglasak pri izradi učinkovitog SAT-rješavača igraju metode za predobradu CNF-formule (engl. *SAT preprocessing*) i metode unutarnje obrade (engl. *SAT inprocessing*)
  - Neki put je diskutabilno može li se predobrada i unutarnja obrada smatrati dijelom samog algoritma za SAT-rješavanje
- Primjer *SAT preprocessing* metoda:
  - PJK, UKČL, ograničena eliminacija varijabli (engl. *bounded variable elimination*), SAT pometanje (engl. *SAT sweeping*)
- Primjer *SAT inprocessing* metoda: rezolucijsko pravilo, proširena rezolucija (engl. *clause-constrained extended resolution*)

Više u: A. Biere, M. Heule, H. Van Marren, T. Walsh, Handbook of Satisfiability, 2nd ed., IOP Press, 2021.

Predobrada i unutarnja obrada

- Ograničena eliminacija varijabli zasnovana je na distribuciji klauzula koja se provodi rezolucijom
  - Izabire se varijabla, dodaju se svi rezolventi te varijable u formulu i uklone se originalne klauzule koje ju sadrže
- Primjer:

$$F = (x \vee e) \wedge (y \vee e) \wedge (\bar{x} \vee z \vee \bar{e}) \wedge (y \vee \bar{e}) \wedge (y \vee z)$$

- Neka za distribuciju klauzula odaberemo varijablu e, rezoluciju na prve dvije klauzule možemo napraviti s trećom i četvrtom, dobivaju se četiri dodatne klauzule:

$$F \wedge (x \vee \bar{x} \vee z) \wedge (x \vee y) \wedge (y \vee \bar{x} \vee z) \wedge (y)$$

- I konačno, miču se klauzule koje sadržavaju e:

$$(y \vee z) \wedge (x \vee \bar{x} \vee z) \wedge (x \vee y) \wedge (y \vee \bar{x} \vee z) \wedge (y)$$

Neki najnoviji uspješni (*state-of-the-art*) SAT-rješavači

- Glucose SAT-solver (2009.+, paralelizacija 2014., L. Simon)
  - Optimizacija procesa upravljanja naučenim klauzulama – agresivnije, dinamičko uklanjanje koje štiti one naučene klauzule koje se češće koriste pri PBO
  - University of Bordeaux - http://www.labri.fr/perso/lsimon/glucose/
- Lingeling, Plingeling, Treengeling (2010.+, A. Biere)
  - Optimizacija niza parametara algoritama, fokus na proučavanju kako najbolje podesiti pretragu iznova, paralelizacija rješavanja SAT-problema
  - Johannes Kepler University of Linz - http://fmv.jku.at/lingeling/
- MapleSAT (i slični) (2016.+, J. H. Liang)
  - Razmatranje grananja kao optimizacijskog problema s ciljem maksimizacije učenja novih klauzula, heuristika LRB je brža od VSIDS-a, ograničena el. varijabli...
  - University of Waterloo i suradnici - https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/
- Rezultati posljednjeg natjecanja SAT-rješavača:
  - https://satcompetition.github.io/2022/results.html

PRIMJENA SAT-RJEŠAVAČA U FORMALNOJ VERIFIKACIJI

Primjena SAT-rješavača u provjeri modela

- SAT-rješavači se u formalnoj verifikaciji najviše koriste u ograničenoj provjeri modela (engl. *bounded model checking,* BMC)
  - Tipično za verifikaciju obilježja sigurnosti: možemo li pronaći loše stanje u *k* koraka?
  - Ili: možemo li pronaći protuprimjer u *k* koraka?
  - Koristi se uglavnom LTL-logika, ali i CTL je podržan
- *k* – koraka: ograničenje na kratak put izvođenja kako bi propozicijska formula bila dovoljno jednostavna za pronaći model
- Problem: što ako se protuprimjer dogodi, ali nakon *k* koraka? **Kod BMC-a, ne otkrije ga se!** Ideja je da se istražuju sve duži i duži putevi, ako se ima dovoljno vremena.

BMC i provjera modela – ideja

- Zadana je Kripkeova struktura M i vremenska formula φ
- Izgradimo formulu Γ(*k*) koja je zadovoljiva akko φ vrijedi na putu duljine *k* stanja
- Put duljine *k* stanja: $I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_{i,} s_{i+1})$
- Za φ = **EF** *p* i za *k* = 2:

$$\Gamma(2) = I(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2) \wedge (p_0 \vee p_1 \vee p_2)$$

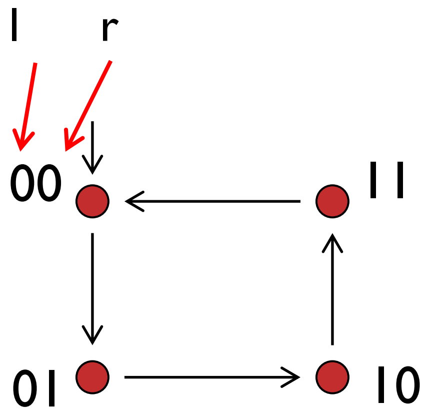
- Za φ = **AG** *p*:

$$\neg\Gamma(k) = (I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_{i,} s_{i+1})) \rightarrow \bigwedge_{i=0}^{k} p_i$$, odnosno

$$\Gamma(k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_{i,} s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p_i$$ Tražimo protuprimjere!

p je očuvan do *k*-tog prijelaza akko je Γ nezadovoljiva

Primjer: Dvobitni brojač




```
MODULE main
VAR
r : boolean;
l : boolean;
```

```
ASSIGN
init(r)  := FALSE;
init(l)  := FALSE;
next(r)  := !r;
next(l)  := r xor l;
```

- Početno stanje: $I : \neg r \wedge \neg l$
- Relacija prijelaza: R: $\begin{pmatrix} l' = (l \neq r) \wedge \\ r' = \neg r \end{pmatrix}$
- Obilježje sigurnosti: **AG** $(\neg l \vee \neg r)$

Ovo se prevede u CNF formu

$$\Gamma(2) : (\neg l_0 \wedge \neg r_0) \wedge \begin{pmatrix} l_1 = (l_0 \neq r_0) \wedge r_1 = \neg r_0 \\ l_2 = (l_1 \neq r_1) \wedge r_2 = \neg r_1 \end{pmatrix} \wedge \begin{pmatrix} (l_0 \wedge r_0) \vee \\ (l_1 \wedge r_1) \vee \\ (l_2 \wedge r_2) \end{pmatrix}$$

$\Gamma(2)$ nije zadovoljiv, ali $\Gamma(3)$ bi bio zadovoljiv, što ne bi pronašli za $k = 2$.

Ulazni format DIMACS CNF

- Standardni ulazni format datoteke za rješavanje SAT-problema
- ASCII format, ekstenzija: .cnf
- *Benchmark* skupovi za SAT: http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html
- Opis formata:

Komentari počinju s "c", obično na početku datoteke

```
c
c start with comments
c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

Problemska linija počinje s "p" , zatim se navodi tip problema (cnf, wcnf, dnf), broj varijabli i na kraju broj klauzula

Klauzule, svaka u svojem retku, završavaju s "0".

Minus ispred broja varijable označava njenu negaciju.

Daljnje učenje – MAX-SAT

- MAX-SAT – problem određivanja najvećeg broja klauzula neke Booleove formule u CNF-obliku koje mogu biti zadovoljive za neko pridruživanje istinitosti
  - NP-težak problem, teži od "običnog" SAT problema
- Značajnija proširenja MAX-SAT problema:
  - Težinski MAX-SAT (engl. *weighted MAX-SAT*) – svaka klauzula ima zadanu težinu – traženje maksimalne težine skupa zadovoljivih klauzula
  - Djelomični MAX-SAT (engl. *partial MAX-SAT*) – za dani podskup klauzula radi se MAX-SAT, ostale klauzule moraju biti SAT
- Natjecanje MAX-SAT-rješavača: https://maxsat-evaluations.github.io/2022/
- Primjer MAX-SAT-rješavača: QMaxSAT – težinski, djelomični MAX-SAT rješavača ne garantira kompletnost – korisno za optimizacijske probleme
  - https://sites.google.com/site/qmaxsat

Daljnje učenje: SMT-rješavači

- Dokazivanje teorija (cjelobrojnih formula, realnih formula, bit-vektorskih formula, kvantifikacija varijabli...) koje su zadane u nekom obliku logike prvog reda s jednakosti
- U nekim slučajevima koriste pretvorbu problema u SAT-oblik pa dalje koriste SAT-rješavače
- Koristi se za automatsku verifikaciju programa (npr. provjeru ugovora, simboličko izvršavanje programa) s ciljem pronalaska pogrešaka, a također i za formalnu sintezu
- Najpoznatiji primjeri SMT-rješavača:
  - Z3: https://github.com/Z3Prover/z3/wiki
  - Yices: http://yices.csl.sri.com/

BDDs vs SAT vs …

- Svi pristupi verifikaciji su komplementarni
- SAT-rješavači ne zamjenjuje BDD-ove, niti QBF-rješavači ili SMT-rješavači ne zamjenjuje SAT-rješavače / BDD-ove
- **"*No silver bullet*" – nema postupka koji će na svim problemima dati najbolji rezultat** – sve ovisi o vrsti problema – neki problemi se bolje rješavaju s BDD-om, a neki sa SAT-om ili SMT-om
- Nažalost, često u literaturi nije unaprijed poznato koji postupak koristiti – treba isprobati više njih
- Nekih pravila ima: BMC sa SAT-om je brži za pronalaženje "plićih" pogrešaka i za davanje kraćih protuprimjera od BDD-ova
- Postupci zasnovani na BDD-ovima su obično bolji od SAT-a za dokazivanje odsutnosti pogreške jer jednostavnije rade nad beskonačnim putovima kroz sustav

Zadaci

1. Pokažite zadovoljivost zadane formule u CNF-obliku korištenjem temeljnog DPLL rješavača. Izbor varijable grananja, ako je on potreban, provedite proizvoljno.

$\Gamma = (x1 \vee \neg x2 \vee x3) \wedge (\neg x1 \vee \neg x2 \vee x3) \wedge (x1) \wedge (x1 \vee \neg x2) \wedge (\neg x2 \vee \neg x4) \wedge (\neg x1 \vee x2 \vee x4)$

2. Imate na raspolaganju SAT-rješavač GRASP.

Za zadani početni skup klauzula K_1-K_9

i za trenutno pridruživanje

$\{x_3=0@1, x_7=0@2, x_{10}=0@2, ...\}$:

a) Nacrtajte graf implikacija ako je trenutna odluka o pridruživanju $x_2 = 1@4$

b) Odredite naučene konfliktne klauzule za otkrivene konflikte na kraju grafova implikacija

c) Odredite jedinstvenu implikacijsku točku za konflikte prouzročene varijablom x_2 te razinu odluke δ na koju će algoritam skočiti nakon konflikata

$K_1 = (\neg x_1 \vee x_5 \vee x_9)$
$K_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_4)$
$K_3 = (x_1 \vee \neg x_2 \vee x_3)$
$K_4 = (x_5 \vee x_2 \vee x_3)$
$K_5 = (\neg x_5 \vee x_8)$
$K_6 = (x_4 \vee x_8)$
$K_7 = (x_7 \vee \neg x_8 \vee \neg x_9)$
$K_8 = (\neg x_1 \vee \neg x_2 \vee \neg x_5)$
$K_9 = (x_2 \vee \neg x_5 \vee \neg x_8)$
...

Zadaci

3. Pokažite zadovoljivost zadanog skupa klauzula korištenjem Chaff-rješavača. Pritom koristite sve značajke algoritma osim uklanjanja naučenih klauzula i pretrage iznova.

$K_1 = (\neg x_1 \vee x_2 \vee x_4)$

$K_2 = (\neg x_1 \vee x_2 \vee \neg x_4)$

$K_3 = (\neg x_1 \vee \neg x_2 \vee x_4)$

$K_4 = (\neg x_1 \vee \neg x_2 \vee \neg x_4)$

$K_5 = (x_1 \vee x_3)$

$K_6 = (x_1 \vee \neg x_3)$